Data Representation Number Systems

Dr. Ahmed El-Bialy
Dr. Sahar Fawzy

Representing Signed Integers

- The number system we studied represent positive numbers
- In decimal, we have positive and negative numbers.
- We should also take into consideration that the number of bits in a binary number is fixed by the hardware design (the machine's word length).

Representing Signed Integers

- There are actually three possible schemes for representing signed numbers.
- All agree on using the leftmost bit in the number to encode the sign
- with 0 encoding +
- and 1 encoding -.
  - **1. Sign magnitude**
  - **2. One's complement**
  - **3. Two's complement**

Sign-Magnitude

- The simplest way to represent negative numbers is called sign-magnitude.
- It is based on the method we use with decimal numbers:
  - To represent a positive number in (say) 8 bits, represent its magnitude as seven bits and prefix a 0.
  - To represent a negative number in (say) 8 bits, represent its magnitude as seven bits and prefix a 1.
- Example: +65 → 01000001
- Example: - 65 → 11000001

1's Complement

- a better method is called 1's complement. In 1's complement, we proceed as follows:
  - To represent a positive number in (say) 8 bits, represent it as an unsigned number using seven bits and prefix a 0.
  - To represent a negative number in (say) 8 bits, represent its absolute value as above, then invert all the bits (which results in a sign bit of 1)
- Example: +65 → 01000001
- Example: - 65 → 10111110

Two's Complement

- To represent a positive number in (say) 8 bits, represent it as an unsigned number using seven bits and prefix a 0.
- To represent a negative number in (say) 8 bits, represent its absolute value as above, then invert all the bits and add 1. (i.e. form its one's complement representation and add 1).

- Example: +65 → 01000001
- Example: - 65 → 10111111

Addition in Binary

- Example:

```
01011010        90
01101100      + 108
----------      ----
11000110        198
```

- Example:

```
00111001        57
01011010      + 90
--------
10010011        147
```

Subtraction in Binary
2'Complement

- Examples:

00001000	8
11111111	-1
--------	--
00000111	7

- Examples:

11111110	-2
11111110	-2
--------	--
1 11111100	-4

carry out of sign position discarded

- Examples:

11000001	-63
01000000	64
--------	--
00000001	1

Real Numbers

- A real number is stored internally as a mantissa times a power of some radix

$$m \times r^{e}$$

- r = 2 (for binary)
- m → mantissa

Conversion of Fraction Decimal to Binary

- Repeat multiplying of the decimal number by 2 and keep the units and replace it with 0:
- e.g.: Convert $(0.40625)_{10}$ to binary

  0.40625 x 2 = 0.8125 units 0
  0.8125 x 2 = 1.625 units 1
  0.625 x 2 = 1.25 unit2 1
  0.25 x 2 = 0.5 unit2 0
  0.5 x 2 = 1.0 unit2 1
  0.0

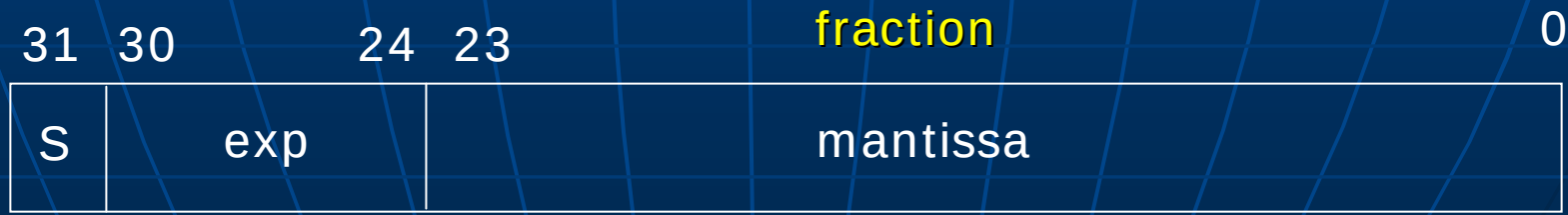
  Now, the trick to remember is that the MSB is ON TOP, so read the answer from the top down:
- $(0.40625)_{10} = (0.01101)_{2}$

IEEE Format

- IEEE has developed a floating point standard (Standard 754)
- The standard provides two different formats for single and double precision numbers.
- We will consider the single precision representation system in some detail.

IEEE Format

- s = sign of mantissa: 0 = +, 1 = -
- exp = exponent as power of 2, stored in excess 127 form : i.e. value stored is 127 + true value.
  - ex: true exponent = 0; stored exponent = 127
  - ex: true exponent = 127; stored exponent = 254
  - ex: true exponent = -126; stored exponent = 1



IEEE Format

- The significand (magnitude of mantissa) is normalized to lie in the range 1.0 <= |m| < 2.0.
- This implies that the leftmost bit is always 1, but need not be stored.
- The 23 bits allocated are used to store the bits to the right of the binary point,
- The missing leftmost bit “1” is inserted to the left of the binary point by the hardware when doing arithmetic. (This is called hidden-bit normalization, and is why this field is labeled "fraction".)

IEEE Format

- e.g.:

$(-5.375)_{10} \rightarrow (-101.011)_2 \rightarrow (-1.01011 \times 2^2)_2$

  - Sign bit =1 (negative)
  - exp = 2 + 127 = 129 = $(10000001)_2$
  - Significand= 1.01011
  - Fraction = 01011
  - IEEE format:

1	10000001	01011000000000000000000

Text

- Computers deals only with numbers
- Therefore, text is coded into numbers.
- There are a variety of character codes in use on various systems, all of which begin by mapping the character set to a set of small unsigned integers, which are then in turn stored in binary form.

Text

- ASCII (American Standard Code for Information Interchange): is the most widely used today.

TABLE 3-4 American Standard Code for Information Interchange (ASCII)

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	−	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

Instructions

- In our study, we will consider the Von Neumann paper that put the ground work for modern computers.
- One key idea was that of using the same memory to store both data and instructions.
- We will be learning more about this topic in this course and next chapters but for now we need to learn more about hardware and logic circuits.

Thank you

see you in the

Logic Circuit and Boolean Algebra